

Collaborative Enforcement of Firewall Policies in Virtual Private Networks

Alex X. Liu Fei Chen
Dept. of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824-1266, U.S.A.
{alexliu, feichen}@cse.msu.edu

ABSTRACT

The widely deployed Virtual Private Network (VPN) technology allows roaming users to build an encrypted tunnel to a VPN server, which henceforth allows roaming users to access some resources as if that computer is residing on their home organization's network. Although the VPN technology is very useful, it imposes security threats to the remote network because their firewall does not know what traffic is flowing inside the VPN tunnel. To address this issue, we propose VGuard, a framework that allows a policy owner and a request owner to collaboratively determine whether the request satisfies the policy without the policy owner knowing the request and the request owner knowing the policy. We first present an efficient protocol, called Xhash, for oblivious comparison, which allows two parties, where each party has a number, to compare whether they have the same number, without disclosing their numbers to each other. Then, we present the VGuard framework that uses Xhash as the basic building block. The basic idea of VGuard is to first convert a firewall policy to non-overlapping numerical rules and then use Xhash to check whether a request matches a rule. Comparing with the Cross-Domain Cooperative Firewall (CDCF) framework, which represents the state-of-the-art, VGuard is not only more secure but also orders of magnitude more efficient. On real-life firewall policies, for processing packets, our experimental results show that VGuard is 552 times faster than CDCF on one party and 5035 times faster than CDCF on the other party.

Categories and Subject Descriptors

C.2.4 [Distributed Systems]: Distributed applications

General Terms

Algorithms, Security

Keywords

virtual private networks, privacy preserving, oblivious comparison, Xhash protocol

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PODC'08, August 18–21, 2008, Toronto, Ontario, Canada.
Copyright 2008 ACM 978-1-59593-989-0/08/08 ...\$5.00.

1. INTRODUCTION

Virtual Private Network (VPN) is a widely deployed technology that allows roaming users to securely use a remote computer on the public Internet as if that computer is residing on their organization's network, which henceforth allows roaming users to access some resources that are only accessible from their organization's network. VPN works in the following manner. Suppose IBM sends a field representative to one of its customers, say Michigan State University (MSU). Assume that MSU's IP addresses range 1.1.0.0 ~ 1.1.255.255 and IBM's IP addresses range 2.2.0.0 ~ 2.2.255.255. To access resources (say a confidential customer database server with IP address 2.2.0.2) that are only accessible within IBM's network, the IBM representative uses an MSU computer (or his laptop) with an MSU IP address (say 1.1.0.10) to establish a secure VPN tunnel to the VPN server (with IP address 2.2.0.1) in IBM's network. Upon establishing the VPN tunnel, the IBM representative's computer is temporarily assigned a virtual IBM IP address (say 2.2.0.25). Using the VPN tunnel, the IBM representative can access any computer on the Internet as if his computer is residing on IBM's network with IP address 2.2.0.25. The payload of each packet inside the VPN tunnel is another packet (to or from the newly assigned IBM IP address 2.2.0.25), which is typically encrypted. Figure 1 illustrates an example packet that traverses from the IBM representative's computer on MSU's network to the customer database server in IBM's network.

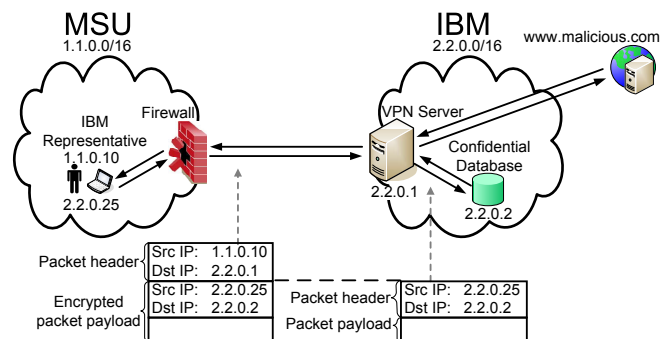


Figure 1: A typical example

While the VPN tunnel is very useful for the IBM representative, it imposes security threats to MSU's network because MSU's firewall does not know what traffic is flowing inside the VPN tunnel. For example, if MSU's firewall blocks access to a remote site (say www.malicious.com) or disallows machines to run peer-to-peer applications due to copyright

concerns, MSU’s firewall cannot enforce its policies on the IBM representative’s computer although that computer is physically on MSU’s network. Basically, the VPN tunnel opens a hole to MSU’s firewall that may allow unwanted traffic to flow inside or outside. Having such a hole is remarkably dangerous because viruses or worms could flood in through it to the IBM representative’s computer first and then further spread to other computers on MSU’s network.

This problem is conceivably difficult to solve for many reasons. First, MSU cannot simply block VPN connections because otherwise the IBM representative may fail to perform his duties. Second, MSU cannot share its firewall policy with IBM. It is common in practice that firewall policies are kept confidential due to security and privacy concerns. Knowing the firewall policy of a network could enable attackers to easily spot the security holes in the policy and launch corresponding attacks. A firewall policy also reveals the IP addresses of important servers, which are usually kept confidential to reduce the chance of being attacked. Furthermore, from a firewall policy one may be able to derive the business relationship of the organization with their partners. Third, IBM cannot share the traffic in its VPN tunnel with MSU due to security and privacy concerns. For example, IBM may want to keep the IP address of its customer database server confidential to reduce the likelihood of being attacked. One main purpose of VPN is to achieve such confidentiality.

The fundamental problem in the above application is: *how can we collaboratively enforce firewall policies in a privacy preserving manner for VPN tunnels in an open distributed environment?* A satisfactory solution to this problem should meet the following three requirements: (1) The request owner cannot gain any more knowledge on the policy after any number of runs of the protocol than they would by brute force probing of the policy. We refer to this requirement as *policy privacy*. (2) It should be computationally infeasible for the policy owner to figure out a request. We refer to this requirement as *request privacy*. (3) The overhead of the solution should be marginal. Timely processing of every request (or packet) is critical for distributed applications. We refer to this requirement as *protocol efficiency*. In addition to the above three requirements on policy privacy, request privacy, and protocol efficiency, a desirable solution should not require a trusted third party; otherwise, the solution will be difficult to deploy. Throughout this paper, we use “MSU” to represent the policy owner and “IBM” to represent the request owner.

Although this is a fundamentally important problem, it is largely underinvestigated. The state-of-the-art on this problem is the seminal work in [3], where Cheng *et al.* proposed a scheme called CDCF. However, CDCF is vulnerable to selective policy updating attacks, by which the policy owner can quickly figure out the request of the other party. Furthermore, CDCF is inefficient because it uses commutative encryption functions (such as the Pohlig-Hellman Exponentiation Cipher [12] and Secure RPC Authentication (SRA) [13]), which are extremely expensive in nature, as the core cryptography primitive.

In this paper, we present VGuard, a secure and efficient framework for collaborative enforcement of firewall policies. In VGuard, different from CDCF, the policy owner does not know which rule matches which request; thus, it makes the selective policy updating attacks infeasible. Furthermore, unlike CDCF, VGuard obfuscates rule decisions, which pre-

vents MSU from knowing the decision for the given packet. To make VGuard efficient, we propose a new oblivious comparison scheme, called Xhash, which uses XOR and secure hash functions. Xhash is three orders of magnitude faster than the commutative encryption scheme used in CDCF. Moreover, VGuard uses decision diagrams to process packets, which is much faster than the linear search used in CDCF. By side by side comparison, our experimental results show that VGuard is hundreds times faster than CDCF in processing packets.

We make the following three key contributions in this paper. First, we propose Xhash, a very efficient oblivious comparison scheme that simply uses XOR and secure hash functions. Second, we propose VGuard, a privacy preserving framework for collaborative enforcement of firewall policies. Third, we implement both VGuard and CDCF and perform extensive experiments to evaluate their performance.

The rest of the paper proceeds as follows. We first introduce our assumptions and threat model in Section 2. We then introduce firewall policies in Section 3. In Section 4, we present our protocol Xhash for oblivious comparison. In Sections 5 and 6, we present the bootstrapping and filtering protocols. We discuss remaining issues in Section 7. We review related work in Section 8. Our experimental results are in Section 9. Finally, we give conclusions in Section 10.

2. ASSUMPTIONS AND THREAT MODEL

Assumptions: First, we assume that the two parties of policy owner and request owner follow the preestablished collaborative enforcement protocol. In particular, the enforcement party does enforce the decision made by the other party. For example, we assume that IBM correctly enforces the decisions made by MSU. This assumption can be realized by the service level agreement between MSU and IBM. Second, we assume that the policy owner may try to figure out the request without violating the protocol and similarly the request owner may try to figure out the policy without violating the protocol. Third, we assume that between two collaborating parties there exists secure channels, which could be achieved using protocols such as SSL.

Threat Model: Without violating the preestablished collaborative enforcement protocol, we assume that the policy owner may attempt to break the request and the request owner may attempt to break the policy.

3. BACKGROUND

We first formally define the concepts of fields, packets, and firewalls. A *field* F_i is a variable of finite length (i.e., of a finite number of bits). The domain of field F_i of w bits, denoted $D(F_i)$, is $[0, 2^w - 1]$. A *packet* over the d fields F_1, \dots, F_d is a d -tuple (p_1, \dots, p_d) where each p_i ($1 \leq i \leq d$) is an element of $D(F_i)$. Firewalls usually check the following five fields: source IP address, destination IP address, source port number, destination port number, and protocol type. The lengths of these packet fields are 32, 32, 16, 16, and 8 respectively. We use Σ to denote the set of all packets over fields F_1, \dots, F_d . It follows that Σ is a finite set and $|\Sigma| = |D(F_1)| \times \dots \times |D(F_d)|$, where $|\Sigma|$ denotes the number of elements in set Σ and $|D(F_i)|$ denotes the number of elements in set $D(F_i)$.

A *rule* has the form $\langle \text{predicate} \rangle \rightarrow \langle \text{decision} \rangle$. A $\langle \text{predicate} \rangle$ defines a set of packets over the fields F_1 through F_d , and is specified as $F_1 \in S_1 \wedge \dots \wedge F_d \in S_d$ where each S_i is a subset of $D(F_i)$ and is specified as either a prefix or a range. A *pre-*

fix $\{0, 1\}^k \{*\}^{w-k}$ with k leading 0s or 1s for a packet field of length w denotes the range $[\{0, 1\}^k \{0\}^{w-k}, \{0, 1\}^k \{1\}^{w-k}]$. For example, prefix 01** denotes the range [0100, 0111]. A rule $F_1 \in S_1 \wedge \dots \wedge F_d \in S_d \rightarrow \langle \text{decision} \rangle$ is a *prefix rule* if and only if each S_i is represented as a prefix. In firewall rules, source IP addresses, destination IP addresses, and protocol types are typically specified as prefixes, and source ports and destination ports are typically specified as ranges. A packet (p_1, \dots, p_d) *matches* a predicate $F_1 \in S_1 \wedge \dots \wedge F_d \in S_d$ and the corresponding rule if and only if the condition $p_1 \in S_1 \wedge \dots \wedge p_d \in S_d$ holds. For firewalls, the typical decisions include permit, deny, permit with logging, and deny with logging. A rule $F_1 \in S_1 \wedge \dots \wedge F_d \in S_d \rightarrow \langle \text{decision} \rangle$ is called a *singleton rule* if and only if $|S_i| = 1$ for every i .

A sequence of rules $\langle r_1, \dots, r_n \rangle$ is *complete* if and only if for any packet p , there is at least one rule in the sequence that p matches. To ensure that a sequence of rules is complete and thus is a firewall, the predicate of the last rule is usually specified as $F_1 \in D(F_1) \wedge \dots \wedge F_d \in D(F_d)$. A *firewall* is a sequence of rules that is complete. Two rules in a firewall may overlap; that is, there exists at least one packet that matches both rules. Furthermore, two rules in a firewall may conflict; that is, the two rules not only overlap but also have different decisions. Firewalls typically resolve conflicts by employing a first-match resolution strategy where the decision for a packet p is the decision of the first (i.e., highest priority) rule that p matches in the firewall. Table 1 shows an example firewall. The format of these rules is based upon the format used in Cisco Access Control Lists.

4. OBLIVIOUS COMPARISON

In this section, we consider the following *oblivious comparison* problem. Suppose we have two parties, denoted MSU and IBM, where each party has a private number, N_1 and N_2 respectively. MSU and IBM want to compare whether $N_1 = N_2$; however, no party wants to disclose its number to the other. In case $N_1 \neq N_2$, no party should learn the value of the other party.

In this paper, we propose a simple and efficient protocol, called Xhash, to achieve oblivious comparison. Xhash works as follows. First, MSU and IBM each choose a secret key K_1 and K_2 respectively. Second, MSU sends $N_1 \oplus K_1$ to IBM. Then, IBM computes $HMAC_k(N_1 \oplus K_1 \oplus K_2)$ and sends the result to MSU. Third, IBM sends $N_2 \oplus K_2$ to MSU. Then, MSU computes $HMAC_k(N_2 \oplus K_2 \oplus K_1)$ and compares it with $HMAC_k(N_1 \oplus K_1 \oplus K_2)$, which was received from IBM. Finally, the condition $N_1 = N_2$ holds if and only if $HMAC_k(N_2 \oplus K_2 \oplus K_1) = HMAC_k(N_1 \oplus K_1 \oplus K_2)$. Figure 2 illustrates the Xhash protocol.

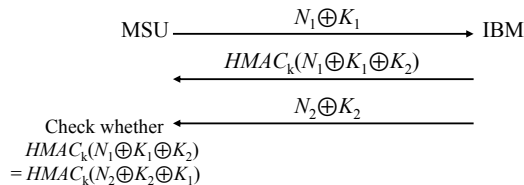


Figure 2: The Xhash protocol

The above function HMAC is a keyed-Hash Message Authentication Code, such as HMAC-MD5 or HMAC-SHA1, which satisfies one-wayness property (i.e., given $HMAC_k(x)$, it is computationally infeasible to compute x and k) and the collision resistance property (i.e., it is computationally

infeasible to find two distinct numbers x and y such that $HMAC_k(x) = HMAC_k(y)$). Note that the key k is shared between MSU and IBM. In this paper, we only consider the cases that N_1 and N_2 are less than 38 bits. Although hash collisions for HMAC do exist in theory, the probability of collision is negligibly small in practice. Furthermore, by properly choosing the shared key k , we can safely assume that HMAC has no collision.

To prevent brute force attacks, we need to choose key K to be sufficiently long. In our implementation, we choose K to be 128 bits. Note that in our framework x is at most 38 bits. To meet the length of K such that x can be XORed with K , we first use a pseudo random generation function R to generate $x_1 = R(x)$. Second, we apply R to x_1 to generate $x_2 = R(x_1)$. Repeat this process until we can concatenate x, x_1, x_2, \dots to form a bit string that meets the length of K . Extra bits in the concatenation beyond the length of K are discarded.

The correctness of Xhash follows from the commutative property of XOR operation (i.e., $x \oplus K_1 \oplus K_2 = x \oplus K_2 \oplus K_1$) and the one-wayness and collision resistance properties of HMAC functions. Note that in the case that $N_1 = N_2$, MSU can compute the secret key K_2 of IBM because $N_2 \oplus K_2 \oplus N_2 = K_2$. However, in applying the above oblivious comparison scheme in VGuard, MSU compares N_2 with many numbers in a set and does not know which number is equal to N_2 .

5. BOOTSTRAPPING PROTOCOL

In the bootstrapping protocol, MSU first converts its firewall policy to a set of non-overlapping prefix rules. Second, MSU converts each prefix to a number. Third, MSU applies XOR operation to every number using its secret key K_1 . Finally, MSU sends the anonymized policy to IBM. Then, IBM further applies XOR and HMAC operations to every number in the received policy using its secret key K_2 , obfuscates the decision of each rule, and shuffle the resulting rules. At last, IBM sends the resulting policy back to MSU.

Converting a firewall policy to a set of non-overlapping prefix rules consists of four steps: FDD construction, range conversion, prefix numericalization, and rule generation.

5.1 FDD Construction

In this step, MSU converts its firewall policy to an equivalent *Firewall Decision Diagram* [6]. A *Firewall Decision Diagram* (FDD) with a decision set DS and over fields F_1, \dots, F_d is an acyclic and directed graph that has the following five properties: (1) There is exactly one node that has no incoming edges. This node is called the *root*. The nodes that have no outgoing edges are called *terminal* nodes. (2) Each node v has a label, denoted $F(v)$. If v is a nonterminal node, then $F(v) \in \{F_1, \dots, F_d\}$. If v is a terminal node, then $F(v) \in DS$. (3) Each edge $e:u \rightarrow v$ is labeled with a nonempty set of integers, denoted $I(e)$, where $I(e)$ is a subset of the domain of u 's label (i.e., $I(e) \subseteq D(F(u))$). (4) A directed path from the root to a terminal node is called a *decision path*. No two nodes on a decision path have the same label. (5) The set of all outgoing edges of a node v , denoted $E(v)$, satisfies the following two conditions: (a) *Consistency*: $I(e) \cap I(e') = \emptyset$ for any two distinct edges e and e' in $E(v)$. (b) *Completeness*: $\bigcup_{e \in E(v)} I(e) = D(F(v))$. Figure 3(a) shows an example firewall policy over two fields F_1 and F_2 , where the domain of each field is $[0, 15]$. The FDD that is semantically equivalent to this firewall policy is

Rule	Source IP	Destination IP	Source Port	Destination Port	Protocol	Action
r_1	1.2.*.*	192.168.0.1	*	25	TCP	accept
r_2	*	*	*	*	*	discard

Table 1: An example firewall

shown in Figure 3(b). Note that in labeling terminal nodes, we use “a” as a shorthand for “accept” (i.e., “permit”) and “d” as a shorthand for “discard” (i.e., “deny”). The algorithm for converting a firewall to an FDD is in [9].

5.2 Range Conversion

For every edge e in the FDD, MSU converts its label $I(e)$ to the minimum set of prefixes whose union is equal to $I(e)$. As one prefix can be converted to one range, a range may have to be converted to multiple prefixes. In converting a range to prefixes, we want to find the minimum set of prefixes such that the union of the prefixes is equal to the range. For example, given range $[0001, 1110]$, the corresponding minimum set of prefixes would be $0001, 001*, 01* *, 10* *, 110*, 1110$. The minimum number of prefixes for representing an integer interval $[a, b]$, where a and b are two numbers of w bits, is at most $2w - 2$ [7]. We call such FDDs, where each edge is labeled by a set of prefixes, *prefix FDDs*. Figure 3(c) shows the prefix FDD converted from the FDD in Figure 3(b).

5.3 Prefix Numericalization

In this step, MSU converts each prefix in the FDD to a concrete number. This process is called *prefix numericalization*. A prefix numericalization function f needs to satisfy the following two properties: (1) for any prefix \mathcal{P} , $f(\mathcal{P})$ is a binary string; (2) for any two prefixes \mathcal{P}_1 and \mathcal{P}_2 , $f(\mathcal{P}_1) = f(\mathcal{P}_2)$ if and only if $\mathcal{P}_1 = \mathcal{P}_2$. There are many ways to do prefix numericalization. We use the following scheme: Given a prefix $b_1b_2 \dots b_k * \dots *$ of w bits, we first replace every $*$ by 0; second, we append $\lceil \log(w + 1) \rceil$ bits whose value is equal to k . For example, $101*$ is converted to 1010011 . After prefix numericalization, the FDD in Figure 3(c) becomes the one in Figure 3(d).

5.4 Applying XOR by MSU

After prefix numericalization, MSU applies XOR to every number in the numericalized FDD using its secret key K_1 . Figure 3(e) shows the numericalized and XORed FDD. Then MSU generates non-overlapping rules from the numericalized and XORed FDD. From each decision path in the FDD, MSU generates a set of non-overlapping rules. For example, from the left-most decision path in Figure 3(e), MSU generates the following four non-overlapping rules:

$$\begin{aligned}
F_1 \in 0100010 \oplus K_1 \wedge F_2 \in 0000010 \oplus K_1 &\rightarrow a, \\
F_1 \in 0100010 \oplus K_1 \wedge F_2 \in 0100011 \oplus K_1 &\rightarrow a, \\
F_1 \in 1000010 \oplus K_1 \wedge F_2 \in 0000010 \oplus K_1 &\rightarrow a, \\
F_1 \in 1000010 \oplus K_1 \wedge F_2 \in 0100011 \oplus K_1 &\rightarrow a,
\end{aligned}$$

Figure 3(f) shows the disjoint rules generated from the FDD in Figure 3(e).

After non-overlapping rules are generated, MSU sends the resulting policy to IBM. If MSU needs to prevent IBM from knowing the number of non-overlapping prefix rules that MSU’s firewall is converted to, MSU can randomly insert some dummy rules formulated by out-of-range dummy numbers and random decisions into the set of non-overlapping numerical rules before applying XOR. An out-of-range dummy number is a number that corresponds to no prefix. For

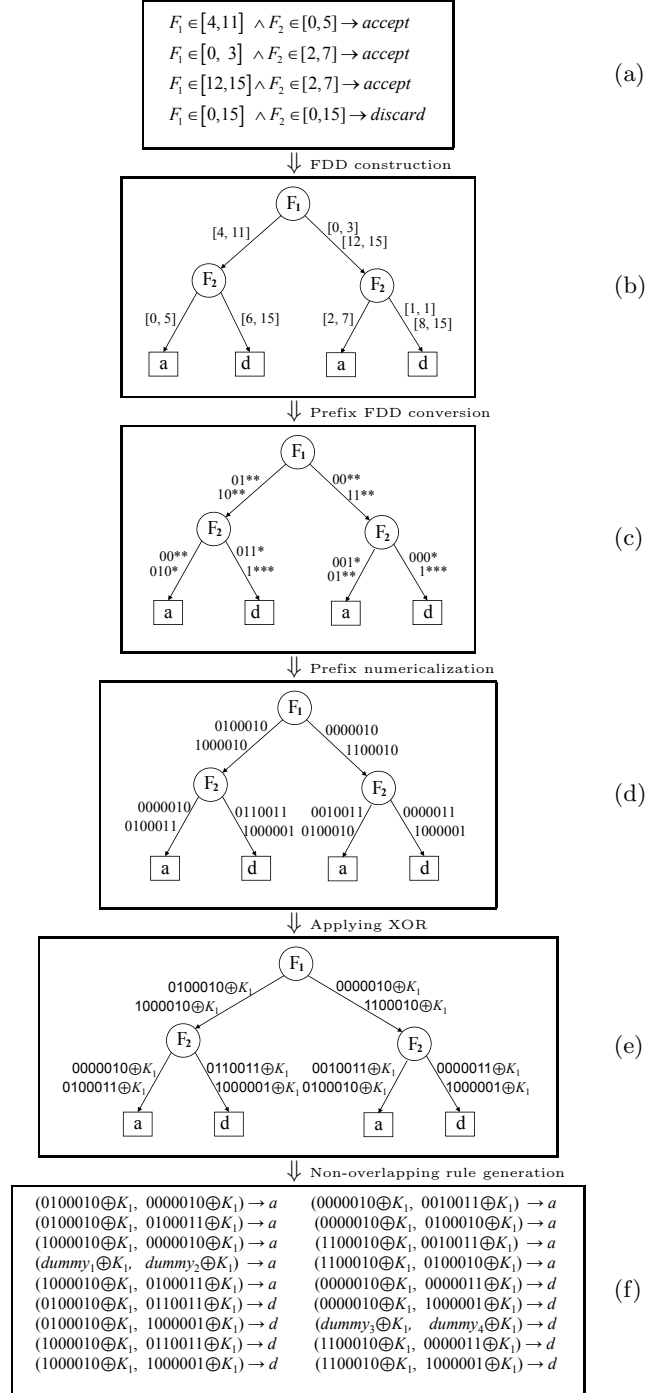


Figure 3: Example of bootstrapping at MSU

example, according to our prefix numericalization scheme, 1011010 is malformed and does not correspond to any prefix. Thus, no packet will match a dummy rule that consists of at least one out-of-range dummy number. Fig 3(f) shows the rules after this step. Note that it is optional for MSU to insert dummy rules. As we will show in the experimental

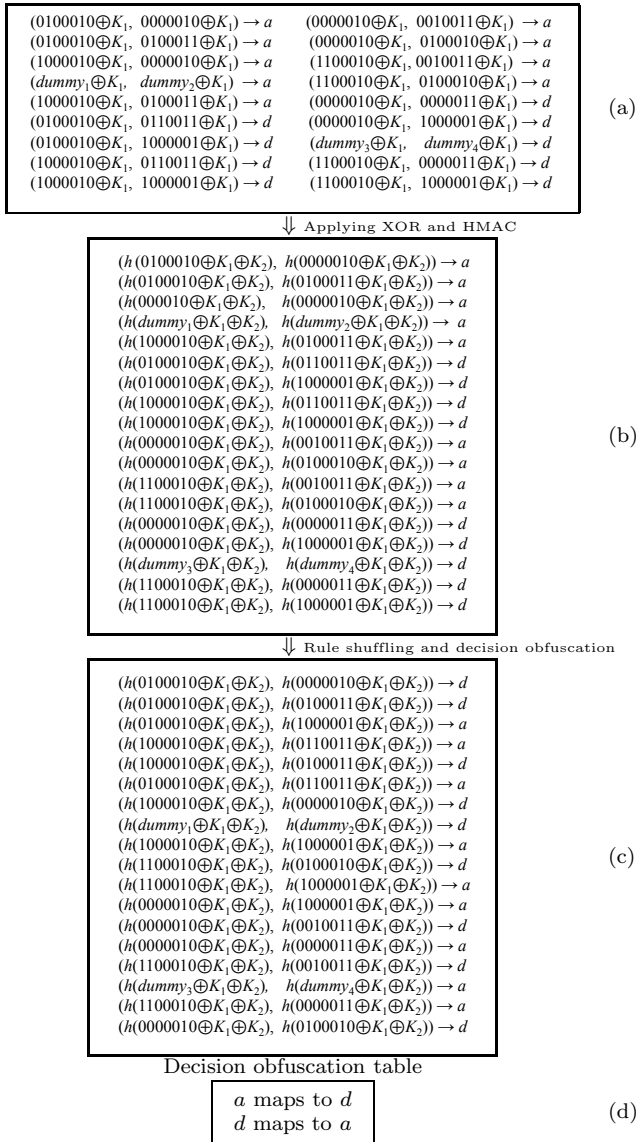


Figure 4: Example of bootstrapping at IBM

results, the number of non-overlapping prefix rules that a firewall is converted to far exceeds the number of original rules. IBM knowing the number of converted rules is much less a concern.

5.5 Applying XOR and HMAC by IBM

Upon receiving a sequence of non-overlapping numerical rules from MSU, IBM further applies XOR and HMAC to every number in the received policy using its secret key K_2 . To destroy the correspondence between the rules after applying XOR and HMAC and the rules received from MSU, IBM randomly shuffles the resulting rules after applying XOR and HMAC. To prevent MSU from knowing the decision of IBM's packet, IBM obfuscates the decision of each rule by mapping each decision to another distinct decision. More formally, the decision obfuscation is a one-to-one mapping function f from the set of all decisions to the same set of all decisions. IBM stores the mapping function f in its decision obfuscation table and replaces the decision of each rule in r_i , say d_i , by $f(d_i)$. To prevent MSU from statistically discovering the obfuscation mapping function f , for any de-

cision d_i , IBM needs to ensure that the number of rules that have decision d_i is the same. This can be easily achieved by adding dummy rules. Due to the rule shuffling and decision obfuscation, MSU cannot correlate the received rules with the original rules, and also cannot identify the decision of each rule. Figure 4(b) shows the rules after IBM applies XOR and HMAC, and Figure 4(c) shows the rules after IBM shuffles rules and obfuscates decisions. The obfuscation mapping function is shown in Figure 4(d). Note that in these figures h denotes the HMAC function. Finally, IBM sends the resulting rules back to MSU.

6. FILTERING PROTOCOL

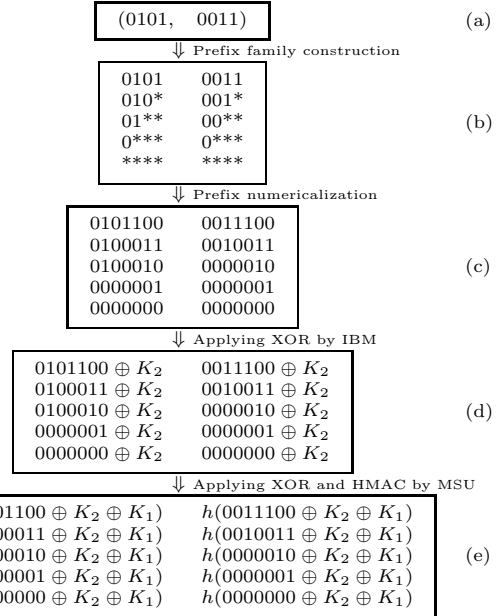


Figure 5: Example of packet preprocessing

In the filtering protocol, each time IBM receives a packet originated from or sent to its representative, IBM first converts the packet to prefixes and then further converts each prefix to a number using the same prefix numericalization scheme. Then, IBM applies XOR to every number in the packet using its secret key K_2 and sends the resulting packet to MSU. MSU further applies XOR and HMAC to the received packet, and searches the obfuscated decision for the packet using the received firewall policy from IBM in the bootstrapping protocol. Finally, MSU sends the obfuscated decision to IBM and IBM finds the original decision using its decision obfuscation table.

6.1 Address Translation

When the IBM VPN server sends (or receives) a packet on behalf of its representative in MSU, the source (or destination) IP address of the packet is an IBM IP address that the IBM VPN server assigned to the IBM representative's computer in MSU. To inquire the decision for this packet from MSU, IBM needs to replace the source (or destination) IP address in the packet by IBM representative's MSU IP address. Otherwise, it is likely that MSU firewall policy blocks all incoming packets that are not sent to MSU and all outgoing packets that are not originated from MSU. Take the example in Figure 1, the packet that IBM should ask MSU for a decision has a source IP address 1.1.0.10 and a destination IP address 2.2.0.2.

6.2 Prefix Membership Verification

We first define two new concepts: k -prefix and prefix family. We call the prefix $\{0, 1\}^k \{*\}^{w-k}$ with k leading 0s and 1s followed by $w - k$ *s a k -prefix. If a value x matches a k -prefix, the first k bits of x and the k -prefix are the same. For example, if $x \in 01**$ (i.e., $x \in [0100, 0111]$), then the first two bits of x must be 01. Given a binary number $b_1b_2 \cdots b_w$ of w bits, the prefix family of this number is the set of $w + 1$ prefixes $\{b_1b_2 \cdots b_w, b_1b_2 \cdots b_{w-1}*, \dots, b_1* \cdots *, ** \dots *\}$, where the i -th prefix is $b_1b_2 \cdots b_{w-i+1} * \cdots *$. We use $PF(x)$ to represent the prefix family of x . For example, $PF(0101) = \{0101, 010*, 01**, 0***, ****\}$. Based on the above definitions, it is easy to draw the following conclusion: given a number x and a prefix \mathcal{P} , $x \in \mathcal{P}$ if and only if $\mathcal{P} \in PF(x)$.

6.3 Packet Preprocessing by IBM

For each of the d fields of a packet, IBM first generates its prefix family. Second, IBM converts each prefix to a number using the same prefix numericalization scheme in the bootstrapping protocol. Third, IBM applies XOR to each number using its secret key K_2 . Last, IBM sends a sequence of d sets of numbers, which corresponds to the d fields of the packet, to MSU. For example, given a packet (0101, 0011) as shown in Figure 5(a), the prefix family of each field is shown in Figure 5(b). The result of prefix numericalization is shown in Figure 5(c). The final 2 sequences of numbers are shown in Figure 5(d).

6.4 Packet Processing by MSU

After MSU receives the packet as d sequences of numbers from IBM, MSU further applies XOR and HMAC using its secret key K_1 . Because of the commutativity property of Xhash, MSU can search the obfuscated decision for the packet using the resulting firewall rules from the bootstrapping protocol. Recall that each rule is represented as d numbers and an obfuscated decision. A packet (s_1, s_2, \dots, s_d) matches a rule $(m_1, m_2, \dots, m_d) \rightarrow \langle obfuscated\ decision \rangle$ if and only if the condition $m_1 \in s_1 \wedge m_2 \in s_2 \wedge \dots \wedge m_d \in s_d$ holds. Therefore, MSU can use linear search to find the first rule that the packet matches. Then, MSU sends the obfuscated decision to IBM and IBM finds the original decision using its decision obfuscation table. Because all the firewall rules resulted from the bootstrapping protocol are non-overlapping, there exists one and only one rule that the packet matches. For example, given the resulting firewall rules in Figure 4(c) and the preprocessed packet in Figure 5(e), the only rule that matches the packet is $(h(0100010 \oplus K_2 \oplus K_1), h(0000010 \oplus K_2 \oplus K_1)) \rightarrow d$.

To improve search efficiency, MSU can use the following two techniques: FDD and hash table. First, MSU converts the non-overlapping rules resulted from the bootstrapping protocol to an equivalent FDD. For example, Figure 6 shows the FDD constructed from the firewall in Figure 4(c). Thus, MSU can search the decision for a packet using the FDD. Second, for the basic operation of testing $m_i \in s_i$, MSU builds d hash tables for the received packet, which is represented as d sequences of numbers.

7. DISCUSSION

7.1 Firewall Updates

When MSU updates its firewall policy, MSU and IBM need to run the bootstrapping protocol again. To prevent

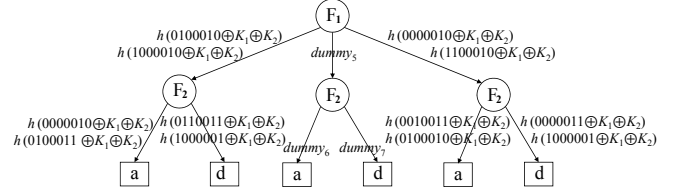


Figure 6: The reconstructed FDD for packet processing

IBM from identifying the unchanged rules, in each run of the bootstrapping protocol, MSU and IBM should change their secret keys K_1 and K_2 . Thus, the rules that IBM receives in each run of the bootstrapping protocol are totally different. Note that MSU and IBM do not need to run the bootstrapping protocol as long as MSU’s firewall remains the same.

7.2 Decision Caching

A packet contains a header (with fields of source and destination IP addresses, source and destination port numbers, and protocol type) and a payload. For different packets with the same header, IBM only needs to ask MSU once and then cache the packet header along with its decision. Whenever the IBM representative builds a connection through the VPN tunnel, IBM first checks whether its cache has an entry that corresponds to the connection. If yes, then IBM executes that decision; if no, IBM asks MSU for the decision using the filtering protocol and then adds an entry into its cache. Because IBM may have multiple collaborators, IBM should record the name of its collaborator (which is “MSU” in this case) to every entry in the cache. Thus, IBM only needs to maintain one cache. When MSU updates its firewall policy and reruns the bootstrapping protocol, IBM needs to delete all the entries that corresponds to MSU. IBM can implement caches efficiently using hash tables or by counting Bloom filters [5].

7.3 Special Treatment of IP Addresses

As we discussed previously, for every packet from (or to) the IBM representative, IBM needs to translate the source (or destination) IP from an address in IBM’s domain to the address in MSU’s domain. Thus, for each packet that IBM asks MSU for a decision, either the source or the destination IP of the packet is the IP that MSU assigns to the IBM representative. This IP address is known to MSU, MSU can use it to compute the secret key K_2 of IBM, which henceforth violates the packet privacy. Note that $x \oplus K_2 \oplus x = K_2$.

To prevent this type of attacks, we modify our framework as follows. First, MSU chooses five secret keys K_1, K_2, \dots, K_5 that correspond to the five packet fields (source IP, destination IP, source port, destination port, and protocol type), and similarly IBM chooses five secret keys K'_1, K'_2, \dots, K'_5 as well. Second, in the bootstrapping protocol, for each non-overlapping rule $(m_1, m_2, \dots, m_5) \rightarrow \langle dec \rangle$ (where each m_i is a number), MSU applies XOR to each m_i using key K_i . Thus, the rule becomes $(m_1 \oplus K_1, m_2 \oplus K_2, \dots, m_5 \oplus K_5) \rightarrow \langle dec \rangle$. Then, MSU sends these rules to IBM. For each rule $(m_1 \oplus K_1, m_2 \oplus K_2, m_3 \oplus K_3, m_4 \oplus K_4, m_5 \oplus K_5) \rightarrow \langle dec \rangle$ that IBM receives from MSU, assuming that m_1 corresponds to the field of source IP and m_2 corresponds to the field of destination IP, IBM creates the following two rules:

$$(m_1 \oplus K_1, HMAC_k(m_2 \oplus K_2 \oplus K'_2), HMAC_k(m_3 \oplus K_3 \oplus K'_3), HMAC_k(m_4 \oplus K_4 \oplus K'_4), HMAC_k(m_5 \oplus K_5 \oplus K'_5)) \rightarrow \langle dec \rangle$$

$$(HMAC_k(m_1 \oplus K_1 \oplus K'_1), m_2 \oplus K_2, HMAC_k(m_3 \oplus K_3 \oplus K'_3), \\ HMAC_k(m_4 \oplus K_4 \oplus K'_4), HMAC_k(m_5 \oplus K_5 \oplus K'_5)) \rightarrow \langle dec \rangle$$

Basically, IBM keeps the source IP field unchanged in the first rule and keeps the destination IP field unchanged in the second rule. At last, IBM sends two sets of rules back to MSU, where in one set the source IP is unchanged and in the other set the destination IP is unchanged. Third, in the filtering protocol, when IBM receives a packet, without loss of generality assuming that the packet is originated from its representative, IBM applies XOR to four fields of the packet (destination IP, source port, destination port, and protocol type) using its four corresponding keys $K'_2, K'_3, K'_4,$ and K'_5 . In other words, when the source IP of the packet is the MSU IP address, IBM does not apply XOR to that field. When the resulting packet $(s_1, s_2, s_3, s_4, s_5)$ is sent to MSU, MSU can easily detect that the source IP field s_1 is unprocessed because MSU knows the IP address of the IBM representative. Then, MSU applies only XOR to s_1 using key K_1 , and process $s_2, s_3, s_4,$ and s_5 as usual using keys K_2, K_3, K_4 and K_5 respectively. Finally, MSU searches the decision for the packet in the rule set where the source IP field was not processed by IBM. Thus, leaving the source (or destination) IP field unprocessed by IBM, MSU cannot break any secret key of IBM.

7.4 Stateful Firewalls

So far we have assumed that firewalls are stateless. A stateful firewall is a firewall that keeps track of the state of network connections across the firewall. When a stateful firewall receives a packet, it first checks its connection table to see whether the packet belongs to an ongoing connection. If yes, the packet is permitted right away. If no, the packet needs to be checked with its stateless rules to determine whether the packet should be permitted; if the stateless rules allows the packet to be permitted, then a new connection is built and inserted into the connection table of the firewall. Such stateful firewalls typically allow inside non-server computers (*i.e.*, the computers that are not servers) to initiate connection with outside computers but disallow outside computers to initiate connection with non-server computers. When an inside non-server computer sends a packet to an outside computer, the stateful firewall uses its stateless rules to decide whether the packet should be permitted; if yes, the firewall adds an entry in its connection table that will allow subsequent packets sent from that outside computer to the inside computer. When an outside computer sends a packet to an insider non-server computer, if there is no corresponding entry in the connection table, the firewall will use its stateless rules to make a decision, which is typically *deny*.

Our framework can be extended to handle stateful firewalls. The basic idea is to let IBM maintain a connection table for its representative. If MSU's firewall is stateful, in the extended framework, when IBM receives a packet from or to its representative, IBM first consults its connection table to see whether the packet belongs to an ongoing connection. If yes, the packet is accepted right away. If no, IBM asks MSU for the decision for this packet; if the packet is permitted, IBM adds an entry into the connection table. Note that the connection table is different from IBM's decision cache. If MSU's firewall is stateless, for every connection, with the help of cache, IBM needs to ask MSU the decision for two packets that go in exactly the opposite direction. If

MSU's firewall is stateful, with the help of the cache and the connection table, IBM only needs to ask MSU the decision for one packet, which is the one that initiates the connection.

7.5 Statistical Analysis Attack and Countermeasures

MSU could launch a statistical analysis attack to reduce the number of possible rules that a packet matches. This attack works as follows. After non-overlapping rule generation, MSU calculates the frequency for each number in the rules. The frequency of each number is preserved in processing the policy by MSU and IBM in the bootstrapping protocol. MSU could exploit such frequency information to reduce the number of possible rules that an IBM packet can match. For example, considering the FDD in Figure 7(a), the frequency of the number 0100010 in the generated non-overlapping rules is 2, and none of the other numbers have the same frequency. Thus, MSU can identify which rules received from IBM correspond to the left branch of the FDD. Interestingly, MSU cannot correlate any rule received from IBM with the right most decision path of this FDD because of the use of dummy rules.

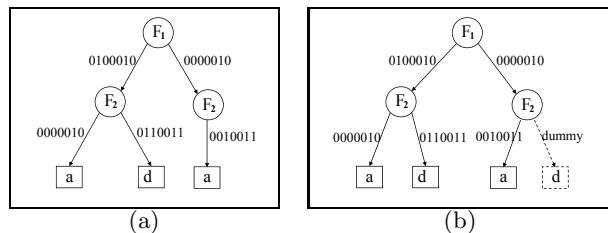


Figure 7: An example of statistical analysis attack

The statistical analysis attack is based on the assumption that the frequency of each number remains the same before and after IBM's processing. Actually, to prevent statistical analysis attacks, IBM can also make a statistical analysis of the hashed rules and add some dummy rules to disturb the statistical properties of the rules. Taking the example FDD in Figure 7(a), IBM can easily destroy the frequency information of the number in the FDD by adding a dummy number to the right F_2 node as shown in Figure 7(b). Basically, IBM generates dummy rules to make all the numbers for each field have the same frequency. This will prevent MSU from launching statistical analysis attacks. Recall that dummy rules use out-of-range numbers and they cannot be matched by any packet.

7.6 Hash Collision

The chance of having hash collisions for HMAC is extremely small. However, to be on the safe side, we propose the following solution to the problem. Our solution is based on the observation that the bootstrapping protocol in our framework can detect hash collisions easily. Recall that the bootstrapping protocol converts firewall policies to non-overlapping rules. If hash collision happens, then among the rules that MSU receives from IBM, there exist at least two rules that are exactly the same. This fact can be used by MSU to easily detect whether hash collision happens. In the case that hash collision does happen, MSU and IBM can simply rerun the bootstrapping protocol, in which they will choose different secret keys and henceforth the hash collision is most likely removed.

8. RELATED WORK

8.1 Secure Function Evaluation

Secure Function Evaluation (SFE) was first introduced by Yao with the famous “Two-Millionaire Problem” [14]. A secure function evaluation protocol enables two parties, one with input x and the other with y , to collaboratively compute a function $f(x, y)$ without disclosing one party’s input to the other. The classical solutions for SFE are Yao’s “garbled circuits” protocol [15] and Goldreich’s protocol [11]. The method provided by Yao has a computational cost of $O(2^b)$, where b is the number of bits needed to encode x and y . Later, the Secure Function Evaluation problem was generalized to the Secure Multiparty Computation (SMC) problem. Chaum *et al.* proved that any multiparty protocol problem can be solved if there is an authenticated secrecy channel between every pair of participants [2]. Zero-knowledge protocols [1, 4, 10] also aims to provide the privacy between two parties. A zero knowledge protocol is an interactive method for one party (suppose IBM) to prove to another (suppose MSU) that a statement is true without revealing anything other than the veracity of the statement. Although we could use SFE or SMC solutions to solve this problem as well as the problem of checking whether a value is in a range, the $O(2^b)$ complexity makes such solutions infeasible.

Li *et al.* proposed Oblivious Attribute Certificates (OACerts) [8]. In OACerts, a credential holder uses his attributes in an oblivious manner to access resources if and only if the attributes in his credential satisfy the server’s policy, and the server does not learn anything about the attribute values of the credential holder, no matter whether the values satisfy the policy or not. Our framework VGuard differs from Li’s scheme in many ways. First, VGuard does not require a trusted third party, while Li’s scheme requires a trusted third party to compute and deliver the certificates for the credential holder and the server. Second, the computational cost of VGuard is much lower than that of Li’s scheme because VGuard is based on efficient XOR and HMAC functions and Li’s scheme is based on expensive PKI operations. Third, the communication cost of VGuard is much lower than that of Li’s scheme because VGuard uses one round of message exchange for processing one packet and Li’s scheme only compares one bit in each round of communication for evaluating *greater or equal* and *less or equal* functions.

8.2 CDCF Framework

Prior work that is closest to ours is the Cross-Domain Cooperative Firewall (CDCF) framework [3]. There are five major differences between VGuard framework and CDCF framework. First, to achieve oblivious comparison, VGuard uses the Xhash protocol and CDCF uses commutative encryption functions (such as the Pohlig-Hellman Exponentiation Cipher [12] and Secure RPC Authentication (SRA) [13]), which are computationally expensive. A commutative encryption function satisfies the following four properties, where K_1 and K_2 are two secret keys: (1) For any x and K , given x and $(x)_K$, it is computationally infeasible to compute the value of K . (2) For any x , K_1 , K_2 , we have $((x)_{K_1})_{K_2} = ((x)_{K_2})_{K_1}$. (3) For any x , y , and K , if $x \neq y$, then we have $(x)_K \neq (y)_K$. (4) For any x and K , given K , $(x)_K$ can be decrypted in polynomial time. Although commutative encryption functions can be used for obli-

ous comparison because of the commutative property, they are computationally expensive. Second, CDCF allows MSU to discover the original firewall rule that a packet matches, which jeopardizes packet privacy. This is particularly dangerous if the matching rule is a singleton rule, which will allow MSU to immediately know the corresponding value in the matching packet. In comparison, VGuard does not allow MSU to discover the original firewall rule that a packet matches. The key operation in VGuard is that it converts the original firewall rules to non-overlapping rules, which enables IBM to shuffle the rules. Thus, MSU cannot figure out the correspondence between the original rules and the received rules from IBM. Because a firewall policy follows first-match semantics, without such a conversion, MSU and IBM can not disturb the order among rules in CDCF. Third, CDCF allows MSU to know the decision for each IBM’s packet, while VGuard does not. Knowing both the original rules and the decision of a packet p , MSU could guess what packets that p could be. Fourth, VGuard uses firewall decision diagrams to speed up the processing of packets, while CDCF uses the straightforward sequential search. Fifth, CDCF does not perform the address translation that we discussed in Section 6.1, which could render MSU’s firewall policy ineffective for IBM’s packets. However, the address translation procedure could be easily added to CDCF.

In particular, CDCF is vulnerable to a type of attacks that we call *selective policy updating attacks*. Such attacks allow MSU to quickly discover the field values in a packet in the following manner. When MSU receives a double encrypted packet p , assuming p matches the prefix rule $F_1 \in S_1 \wedge \dots \wedge F_d \in S_d \rightarrow \langle decision \rangle$, MSU splits each prefix S_i into two prefixes by instantiating the first $*$ by 0 and 1 respectively, and therefore converts the rule to a maximum of 2^d rules. For example, suppose a packet p from IBM matches the prefix rule $F_1 \in 10** \wedge F_2 \in 001* \rightarrow a$. Then, MSU splits the rule into the following four rules: $F_1 \in 100* \wedge F_2 \in 0010 \rightarrow a$, $F_1 \in 101* \wedge F_2 \in 0010 \rightarrow a$, $F_1 \in 100* \wedge F_2 \in 0011 \rightarrow a$, $F_1 \in 101* \wedge F_2 \in 0011 \rightarrow a$. Then, MSU requests to rerun the CDCF protocol due to firewall update. In the new run of CDCF, MSU replaces the above rule by the rules after splitting. (Note that d is typically 4 or 5.) After MSU receives the double encrypted rules from IBM in the new run, MSU compares p with the split rules again. One of the split rules must match p . The above process repeats with a maximum of 32 times before p matches a singleton rule at the end. To counter the selective policy updating attacks, CDCF can be fixed by updating the secret keys on both MSU and IBM sides in each run of the CDCF protocol.

9. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of VGuard on both real-life and synthetic firewall policies. In particular, we compare VGuard and CDCF side by side. We implemented both VGuard and CDCF using Java 1.6.3. Our experiments were carried out on a desktop PC running Windows XP SP2 with 3G memory and dual 3.4 GHz Intel Pentium processors. On real-life firewall policies, for processing packets, our experimental results show that VGuard is 552 times faster than CDCF on MSU side and 5035 times faster than CDCF on IBM side. On synthetic firewall policies, for processing packets, our experimental results show that VGuard is 252 times faster than CDCF on MSU side and 5529 times faster than CDCF on IBM side.

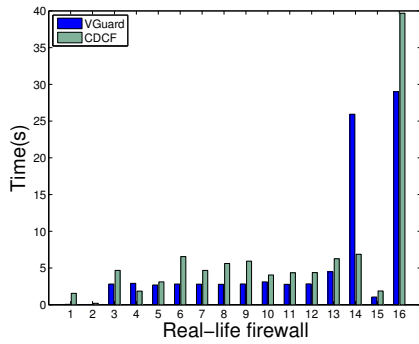


Figure 8: The bootstrapping time of MSU on real-life firewalls

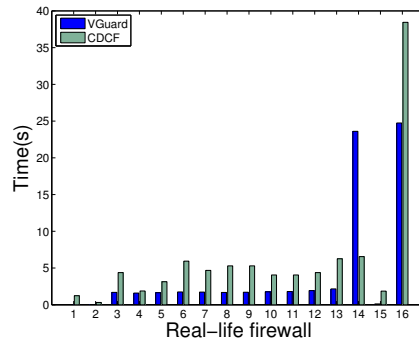


Figure 9: The bootstrapping time of IBM on real-life firewalls

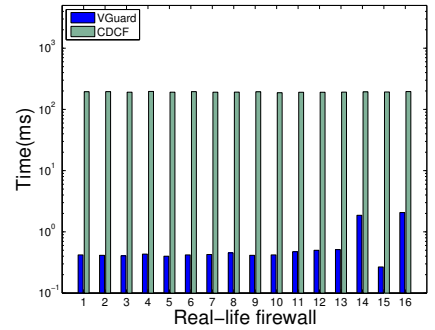


Figure 10: The filtering time of MSU on real-life firewalls

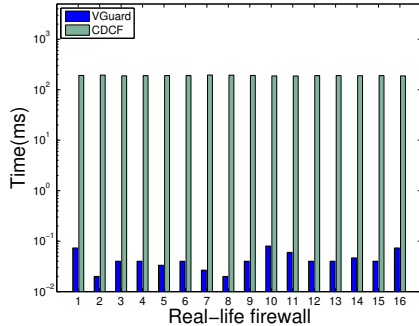


Figure 11: The filtering time of IBM on real-life firewalls

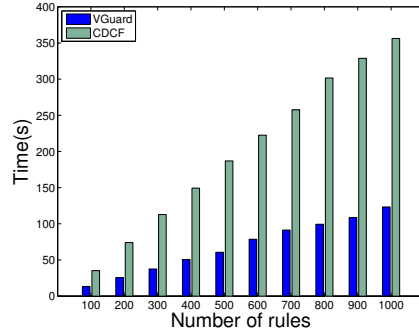


Figure 12: The bootstrapping time of MSU on synthetic firewalls

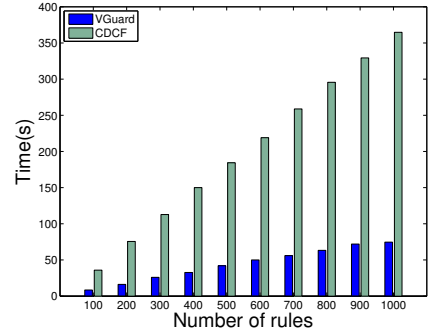


Figure 13: The bootstrapping time of IBM on synthetic firewalls

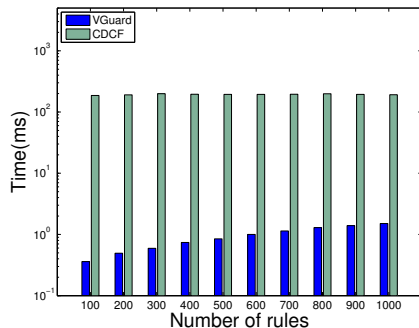


Figure 14: The filtering time of MSU on synthetic firewalls

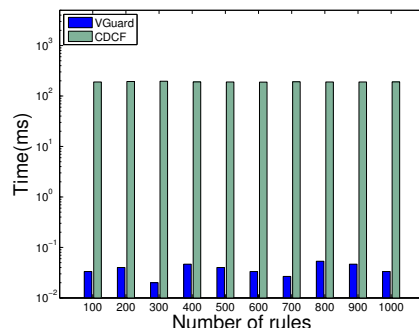


Figure 15: The filtering time in IBM

9.1 Efficiency on Real-life Firewall Policies

We conducted experiments on a set of 16 real-life firewall policies that we collected from a variety of sources. Each firewall examines five packet fields of source IP, destination IP, source port, destination port, and protocol type. The number of rules ranges from dozens to hundreds. We measured the computational cost of the two parties MSU and IBM for both bootstrapping and filtering protocols. For fair comparison, when we implemented CDCF, we used the same parameters as in [3]; for example, we used the Pohlig-Hellman algorithm [12] with a 1024-bit prime modulus and 160-bit encryption keys. In implementing VGuard, we chose the HMAC-MD5 hash function with 128-bit keys.

Figures 8 and 9 show the computational cost of MSU and IBM respectively in the bootstrapping protocol for both VGuard and CDCF. From these two figures, we observe that the bootstrapping cost of VGuard is lower than that of CDCF for most firewalls. Although the Xhash scheme is three orders of magnitude faster than the commutative

encryption scheme, the bootstrapping cost of VGuard is not three orders of magnitude lower than CDCF because VGuard converts the given firewall policy to non-overlapping prefix rules, which results in a significant expansion. Note that the bootstrapping protocol only needs to run once between MSU and IBM unless MSU updates its firewall policy. The performance of the bootstrapping protocol is less critical than that of the filtering protocol.

Figures 10 and 11 show the computational cost of MSU and IBM respectively in the filtering protocol for both VGuard and CDCF. Note that the vertical axis of these two figures are in a logarithmic scale. These two figures show that the filtering cost of VGuard is significantly lower than that of CDCF. On average, VGuard is 552 times faster than CDCF on MSU side and 5035 times faster than CDCF on IBM side. Note that the packet processing time for CDCF on both MSU and IBM side in these two figures seems constant, instead of increasing as the number of rules increases. This is because in processing each packet on both MSU and

IBM side, for CDCF, the encryption time, which is roughly constant for each packet, is about 20 times more than the time for performing a linear search in the firewall.

9.2 Efficiency on Synthetic Firewall Policies

Firewall policies are considered confidential due to security concerns. It is difficult to get a large number of real-life firewall policies to experiment with. To further evaluate the performance of VGuard in comparison with CDCF, we generated a large number of synthetic firewall policies and conducted experiments on them. Every predicate of a rule in our synthetic firewall has five fields: source IP address, destination IP address, source port number, destination port number, and protocol type. We first randomly generated a list of values for each field. For IP addresses, we generated a random class C address then generated single IP addresses within the class C addresses; for ports we generated a random range; for protocols, we choose either TCP, UDP, or ICMP. Every field also has the "*" value included in the list. We then generated a list of predicates by taking the cross product of these five lists and randomly selected from the cross product until we reached our desired classifier size by including a final default predicate. Finally, we randomly assigned one of two decisions, accept or discard, to each predicate to make a complete rule. We generated firewall policies with the number of rules ranging from 100 to 1000, where for each number we generated ten synthetic firewall policies.

Figures 12 and 13 show the computational cost of MSU and IBM respectively in the bootstrapping protocol for both VGuard and CDCF. Figures 14 and 15 show the computational cost of MSU and IBM respectively in the filtering protocol for both VGuard and CDCF. On average, for these synthetic firewall policies, VGuard is 252 times faster than CDCF on the MSU side and 5529 times faster than CDCF on the IBM side.

10. CONCLUDING REMARKS

In this paper, we propose VGuard, a privacy preserving framework for collaborative enforcement of firewall policies. In terms of security, comparing with the state-of-the-art CDCF scheme, VGuard is more secure because of two major reasons. First, VGuard converts a firewall policy of an ordered list of overlapping rules to an equivalent non-ordered set of non-overlapping rules, which enables rule shuffling and consequently MSU cannot identify which original rule matches the given packet. Second, VGuard obfuscates rule decisions, which prevents MSU from knowing the decision for the given packet. In terms of efficiency, comparing with the state-of-the-art CDCF scheme, VGuard is hundreds of times faster than CDCF in processing packets because of two reasons. First, VGuard uses a new oblivious comparison scheme proposed in this paper, which is three orders of magnitude faster than the commutative encryption scheme used in CDCF. Second, VGuard uses firewall decision diagrams for processing packets, which is much faster than the linear search used in CDCF. We want to emphasize that the VGuard framework can be applied to other types of security policies as well. It is also worth noting that the Xhash scheme can be used for other applications that require oblivious comparison.

Acknowledgement

The authors would like to thank Jerry Cheng, Hao Yang, and anonymous referees for their valuable comments. The

work is supported by MSU IRGP Grant and the National Science Foundation under Grant No. CNS-0716407.

11. REFERENCES

- [1] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *Proceedings of the Advances in Cryptology (EUROCRYPT), volume 1807 of Lecture Notes in Computer Science*, May 2000.
- [2] David Chaum, Claude Crepeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 11–19, 1988.
- [3] Jerry Cheng, Hao Yang, Starsky H.Y. Wong, and Songwu Lu. Design and implementation of cross-domain cooperative firewall. In *Proceedings of the IEEE International Conference on Network Protocols (ICNP) '2007*, 2007.
- [4] Ronald Cramer, Matthew K. Franklin, Berry Schoenmarks, and Moti Yung. Multi-authority secret-ballot elections with linear work. In *Proceedings of the Advances in Cryptology (EUROCRYPT)*, 1996.
- [5] Li Fan, Pei Cao, Jussara Almeida, and Andrei Broder. Summary cache: A scalable wide-area web cache sharing protocol. In *Proceedings of the ACM SIGCOMM*, September 1998.
- [6] Mohamed G. Gouda and Alex X. Liu. Structured firewall design. *Computer Networks Journal (Elsevier)*, 51(4):1106–1120, March 2007.
- [7] Pankaj Gupta and Nick McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, 2001.
- [8] Jiangtao Li and Ninghui Li. Oacerts: Oblivious attribute certificates. In *Proceedings of the 3rd Conference on Applied Cryptography and Network Security (ACNS)*, pages 301–317, June 2005.
- [9] Alex X. Liu and Mohamed G. Gouda. Diverse firewall design. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN-04)*, pages 595–604, June 2004.
- [10] Wenbo Mao. Guaranteed correct sharing of integer factorization with off-line shareholders. In *Proceedings of the Public Key Cryptography (PKC), volume 1431 of Lecture Notes in Computer Science*, February 1998.
- [11] Silvio Micali Oded Goldreich and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM Conference on Theory of computing*, May 1987.
- [12] Stephen C. Pohlig and Martin E. Hellman. An improved algorithm for computing logarithms over $gf(p)$ and its cryptographic significance. *IEEE Transactions Information and System Security*, IT-24:106–110, 1978.
- [13] David K. Hess David R. Safford and Douglas Lee Schales. Secure RPC authentication (SRA) for TELNET and FTP. Technical report, 1993.
- [14] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 160–164, 1982.
- [15] Andrew C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986.